ELSEVIER

Discrete Mathematics 226 (2001) 51–70

DISCRETE MATHEMATICS

www.elsevier.com/locate/disc

Optimal binary linear codes of dimension at most seven

Iliya Bouyukliev[a, 1], David B. Jaffe[b, *, 2]

[a] *Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, P.O. Box 323, 5000 Veliko Tarnovo, Bulgaria*

[b] *Department of Mathematics and Statistics, University of Nebraska, Lincoln, NE 68588-0323, USA*

Received 6 July 1999; revised 20 March 2000; accepted 3 April 2000

Abstract

We classify optimal $[n,k,d]$ binary linear codes of dimension $\leqslant 7$, with one exception, where by *optimal* we mean that no $[n-1,k,d]$, $[n+1,k+1,d]$, or $[n+1,k,d+1]$ code exists. In particular, we present (new) classification results for codes with parameters $[40,7,18]$, $[43,7,20]$, $[59,7,28]$, $[75,7,36]$, $[79,7,38]$, $[82,7,40]$, $[87,7,42]$, and $[90,7,44]$. These classifications are accomplished with the aid of the first author's computer program `Extension` for extending from residual codes, and the second author's program `Split`. © 2001 Published by Elsevier Science B.V. All rights reserved.

1. Introduction

A *binary linear code* (or simply a *code*, in this paper) is a subspace of $\mathbb{F}_2^n$ for some n, which is called the *length* of the code. Two codes are *isomorphic* if they have the same length and one may be converted to the other via a permutation of coordinates.

The parameters $[n,k,d]$ of a code are its length n, its dimension k, and its minimum distance d. The most interesting and useful codes are those whose parameters are in some sense extremal. More specifically, an $[n,k,d]$ code is *distance-optimal* if no $[n,k,d-1]$ code exists. It is *length-optimal* (which is stronger) if no $[n-1,k,d]$ code exists. Finally, we say that a code is *optimal* if it is length-optimal and moreover no $[n+1,k+1,d]$ or $[n+1,k,d+1]$ code exists. Such a code cannot be obtained by shortening or puncturing any other binary linear code.

Regardless of which version of optimal one uses, the study of optimal codes may be subdivided into three parts: constructing optimal codes, proving the nonexistence of codes, and the classification of codes. These three parts are intertwined, yet each has its own distinct character and methods.

* Corresponding author.

E-mail addresses: lpmivt@vt.bia-bg.com (I. Bouyukliev), jaffe@genome.wi.mit.edu (D.B. Jaffe).

[1] Partially supported by the Bulgarian National Science Fund under Contracts No. MM-502/1995.

[2] Partially supported by the US National Science Foundation.

0012-365X/01/$ - see front matter © 2001 Published by Elsevier Science B.V. All rights reserved.
PII: S0012-365X(00)00125-4

This paper classifies some codes, so we will say a bit about broader points of view on how this may be done, without making a serious attempt to survey the literature.

There are specialized methods which permit the classification of certain special types of codes. In particular, codes for which d is unusually large (at the Griesmer bound) have been classified (in principle) by van Tilborg [22] and Helleseth [10]; cf. [8,6]. Self-dual codes (see e.g. [19]) and codes having only two nonzero weights (see e.g. [4,5,12]) also admit special methods of classification.

Beyond this, the best classification methods almost inevitably utilize the following theme. To classify $[n,k,d]$ codes, you first classify some codes with smaller n and/or k, and then you build all the $[n,k,d]$ codes by some kind of search procedure from the first batch of codes, testing isomorphism and/or computing isomorphism groups of codes as the search progresses. The whole procedure is quite complex.

Some a priori knowledge about the codes can expedite the search. In particular, if one knows that the weights of the codes lie in a restricted set, then the search will be faster. Linear programming methods (which are also used in proving nonexistence of codes) can be used to restrict weights.

In this paper, we try to classify $[n,k,d]$ optimal codes having $k \leqslant 7$ and $n \leqslant 2^k$. We proceed by classifying residual codes with respect to a word of weight d, which have parameters $[n-d,k,d/2]$, following an algorithm due to the first author, implemented by him as the program `Extension`. This and the second author's program `Split` are used to complete the work.

With the exception of codes with parameters $[56,7,26]$, we complete this classification. Included herein are the classifications of codes with parameters $[40,7,18]$, $[43,7,20]$, $[59,7,28]$, $[75,7,36]$, $[79,7,38]$, $[82,7,40]$, $[87,7,42]$, and $[90,7,44]$, all of which are new. Moreover, the completed classification reveals certain behavior (see Section 2) which may hold for higher-dimensional codes.

We note that the *existence* problem for length-optimal seven-dimensional codes was solved by van Tilborg [23] in 1981. The corresponding existence problem for eight-dimensional codes was recently completed [2]. Before that we could not say which parameters corresponded to *optimal* seven-dimensional codes.

Remark. Obviously, optimal codes with $d > 2^k$ are not projective. Their length is relatively large and it is difficult to investigate them using the methods of this paper. Some classes of codes meeting the Griesmer bound have been classified theoretically, including codes with $d > 2^k$ (see for example the generalized MacDonald codes [7,9]).

2. Observations from the data

In searching for optimal codes, three somewhat vague statements emerge. First, optimal codes tend not to have words of odd weight. Second, the duals of optimal codes

tend to have a small number of words of weight two. Third, optimal codes tend not to have words of large weight.

As for the first assertion, we note that for the parameters under consideration, only three codes have odd weights: [87_7_42.a10], [87_7_42.a19], and [90_7_44.a4]. (These notations refer to certain specific codes, e.g. the first one refers to certain $[87,7,42]$ code named [a10]: see Section 7.) Beyond this summary, we are not sure what generalization might lie.

As for the second assertion, we refer to the tables in Section 4, which show the maximum number of words of weight two which can (and do) occur in the dual codes of optimal codes.

We note that there is a connection between the second and third assertions, as follows. If a k-dimensional code has no dual words of weight two, then it can have no word of weight greater than 2^{k-1}. More generally, if it has at most r dual words of weight two, then it can have no word of weight greater than $2^{k-1}+r$.

Now we proceed to a precise statement about the third assertion, which applies to a somewhat broader class than the optimal codes.

Theorem. *Let C be an $[n,k,d]$ binary linear code which is length-optimal, for which d is even, and such that $n\leqslant 2^k$. Assume that $2\leqslant k\leqslant 7$. Let w be a word in C whose weight exceeds 2^{k-1}. Then $|w|=2^{k-1}+4$, $k\geqslant 6$, and C has parameters $[2^{k-1}+k+4,k,2^{k-2}+4]$.*

This theorem is established by a case-by-case analysis of all possible parameter choices.[3] Of course we would much prefer a completely different sort of proof!

With great trepidation, one might conjecture that the theorem also holds without the hypothesis that $k\leqslant 7$. It seems more likely that some modest adjustments are needed to arrive at a correct general statement.

We note that the hypothesis $n\leqslant 2^k$ is essential. Indeed for n slightly larger than 2^k, d itself exceeds 2^{k-1}. It is also not clear what sort of theorem could hold if length-optimality were replaced by the weaker condition of distance-optimality. For example, $[29,5,14]$ codes are distance-optimal but can have words of weights 17, 18, 20, and 22.

One line of attack on the theorem would be to try to show that if a short code has a long word (i.e. a word whose length is relatively close to n), then there is a bit which is disjoint from all its minimum distance codewords. This approach does not look hopeful, because for example there is a $[29,5,14]$ code, having a word of weight 22, whose minimum distance codewords cover every bit. So perhaps one should aim for a result which asserts that if a short code has a long word, then after an appropriate

[3] The statement refers to length-optimal codes, and this paper is about *optimal* codes. The missing details (all minor) are left to the reader.

modification to the code (what?), it has a bit which is disjoint from all the minimum distance codewords.

We note that the existence of long codewords is related to the existence of disjoint codewords. Amongst the optimal codes of dimension $\leqslant 7$, and length $\leqslant 2^k$, the latter occurs only for the following codes: the unique $[8,4,4]$, $[16,5,8]$, $[32,6,16]$, and $[64,7,32]$ codes, $[24,7,10]$ (code `[a]`), and $[56,7,26]$ (codes described later, named `[disjoint1]` ,..., `[disjoint18]`).

3. An interesting series of codes

In this section we consider codes with parameters $[2^{k-1}+k+4, k, 2^{k-2}+4]$ having a word of weight $2^{k-1}+4$, where $k \geqslant 2$. As already indicated, some of these are length-optimal codes with an unusually large weight. We would like to classify the codes, regardless of whether they are length-optimal. The following tables summarize the state of our knowledge:

k	$[n,k,d]$	No. of codes
2	$[8,2,5]$	1
3	$[11,3,6]$	1
4	$[16,4,8]$	1
5	$[25,5,12]$	1

k	$[n,k,d]$	No. of codes
6	$[42,6,20]$	2
7	$[75,7,36]$	6
8	$[140,8,68]$	(None known)
9	$[269,9,132]$	(None known)

The `Split` code which establishes the assertions of this table may be found in Section 9.

4. Summary of the codes

In this section we give tables which summarize the optimal codes of dimension $k \leqslant 7$ and length $\leqslant 2^k$.

For those codes which are unique, the fact that this is the case is usually a consequence of much earlier results by van Tilborg [22] and Helleseth [10] regarding the classification of codes at the Griesmer bound. For more details, we refer to those papers and to the quick survey in [11]. The codes themselves are due to Solomon and Stiffler [20] and Belov [1].

The classification of $[21,5,10]$ codes is due to van Tilborg [22]. The classification of $[24,7,10]$ codes is due to Kapralov [16] and Jaffe [13] (independently). The uniqueness of $[27,7,12]$ codes is due to Brouwer [3].

The number shown for '$\max \mu_2$' is the maximum number of words of weight two which can (and does) occur in the dual of a code with the given parameters. (If blank, the number is zero.) This is known in all cases expect that of $[56,7,26]$ codes.

$[n,k,d]$	No. of codes	max μ_2	Possible nonzero weights
[8, 4, 4]	1	—	{4, 8}
[12, 4, 6]	1	—	{6, 8}
[16, 5, 8]	1	—	{8, 16}
[21, 5, 10]	2	—	{10, 12, 14, 16}
[24, 5, 12]	1	—	{12, 16}
[28, 5, 14]	1	—	{14, 16}

$[n,k,d]$	No. of codes	max μ_2	Possible nonzero weights
[32, 6, 16]	1	—	{16, 32}
[38, 6, 18]	1	—	{18, 20, 22, 32}
[45, 6, 22]	1	—	{22, 24, 30}
[48, 6, 24]	1	—	{24, 32}
[53, 6, 26]	2	—	{26, 28, 30, 32}
[56, 6, 28]	1	—	{28, 32}
[60, 6, 30]	1	—	{30, 32}

$[n,k,d]$	No. of codes	max μ_2	Possible nonzero weights
[24, 7, 10]	6	—	{10, 12, 14, 16, 18, 22}
[27, 7, 12]	1	—	{12, 16, 20}
[40, 7, 18]	172	1	{18, 20, 22, 24, 26, 28, 30, 32, 34}
[43, 7, 20]	7	—	{20, 24, 28, 32, 36}
[56, 7, 26]	> 19000	4–9	{26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56}
[59, 7, 28]	143	4	{28, 32, 36, 40, 44, 48, 52}
[64, 7, 32]	1	—	{32, 64}
[71, 7, 34]	1	—	{34, 36, 38, 64}
[75, 7, 36]	3606	9	{36, 38, 40, 42, 44, 48, 52, 56, 60, 64, 68}
[79, 7, 38]	216	9	{38, 40, 42, 44, 46, 48, 54, 56, 62, 64}
[82, 7, 40]	11	9	{40, 44, 48, 56, 64}
[87, 7, 42]	55	3	{42, 43, 44, 45, 46, 48, 50, 52, 54, 58, 62, 64}
[90, 7, 44]	6	3	{44, 45, 46, 48, 52, 60, 64}
[93, 7, 46]	1	—	{46, 48, 62}
[96, 7, 48]	1	—	{48, 64}
[102, 7, 50]	3	—	{50, 52, 54, 56, 58, 60, 64}
[105, 7, 52]	1	—	{52, 56, 60}
[109, 7, 54]	1	—	{54, 56, 62, 64}
[112, 7, 56]	1	—	{56, 64}
[117, 7, 58]	2	—	{58, 60, 62, 64}
[102, 7, 60]	1	—	{60, 64}
[124, 7, 62]	1	—	{62, 64}

5. Notes on the computations

There are two general methods which enter into the proofs of the results in this paper. One such method involves a new algorithm for extension from residual codes, implemented as the first author's program `Extension`. The other method involves linear programming, based on various weight enumerators. Both methods have been implemented in the second author's program `Split`, and we present in this paper a program, in the `Split` language, which confirms our claims. [4] Thus, the calculations involving extension from residual codes have been verified using two independently written software systems.

[4] These calculations make extensive use of Brendan McKay's graph-isomorphism program `nauty`. We refer to McKay [17,18] for more information about the latter.

The calculations were carried out on a 500 MHz Alpha 21264 processor. The `Unresidue` line in the $[75,7,36]$ computation took about 18 h. The corresponding line in the $[87,7,42]$ computation took about 5 h. The construction of 19549 $[56,7,26]$ codes took about 14 h. All the other computations amounted in total to about 6 h.

Because these computations must be carried out to even define the codes, we note the following conveniences.

- All the codes are accessible over the web — see [14].
- The commands which define the codes (mostly those containing :=) will (upon execution) cache the code definitions. Upon a second execution, if the line '`set gullible`'; has been added, `Split` will draw the definitions from the cache, rather than recompute them from scratch.
- A list of the weight enumerators occurring amongst the codes[5] is attached as an appendix.
- A file containing the generator matrices of the codes constructed here is available as an 'electronic appendix' to this paper.

6. Main ideas of extension

Let C_x be $[n,k,d]$ codes and C_0 be their residual code $\mathrm{Res}(C_x,w)$ with respect to a codeword of weight $w<2d$. In this section we describe how to construct all the $[n,k,d]$ codes C_x if we know a residual code C_0.

Let us denote a generator matrix of the code C_0 by G_0. Then there exist generator matrices of C_x in the following form:

$$G_x=\left(\begin{array}{c|c} 1\ 1\ \cdots\ 1 & 0\ \cdots\ 0\ 0 \\ \hline \cdots\cdots\cdots & \\ \cdots\cdots\cdots & G_0 \\ & \\ \cdots\cdots\cdots & \end{array}\right).$$

We suggest a backtrack algorithm for finding all solutions for the second row (having fixed the first) and all solutions for the row $t+1$ (having fixed the first t rows). Our basic information will be the number and length of the intervals in which the columns (between 1 and w) of the matrix of the first t rows are the same. We will call a solution in the $(t+1)$th step the number of 1's in any interval for a possible $(t+1)$th row of the generator matrix.

Let us denote the possibilities for the number of 1's in position $1\ldots w$ by x_2^1 for the second row, by x_3^1 for the third row and x_k^1 for the last row.

It is not very difficult to find using exhaustive search all the values of the variables depending on W (where W is the set of possible nonzero weights) and G_0. Let us

[5] The 174 enumerators of $[75,7,36]$ codes are not included.

denote all the values of the solutions by $\{x_2^1\},\{x_3^1\},\ldots,\{x_k^1\}$. Actually, in the beginning we have only one interval.

Let $z \in \{x_2^1\}$. We may put z 1's in positions $1 \ldots z$ in the second row without loss of generality.

Then some of the codes C_x (if such codes exist for z) will have generator matrices in the following form:

$$G'_x = \left(\begin{array}{cccc|c} 1 & 1 & \ldots & 1 & 1 \ldots 1 & 0 \ldots 0 \\ 1 & 1 & \ldots & 1 & 0 \ldots 0 & \\ & \ldots & & \ldots & & G_0 \\ & \ldots & & \ldots & & \end{array}\right).$$

We denote the number of 1's in positions $1 \ldots z$ and $z+1 \ldots w$ in the third row by $x_{3_1}^2, x_{3_2}^2$ and in row k by $x_{k_1}^2, x_{k_2}^2$. The upper index means that we have two fixed rows and the other indices fix the row and the interval. We have the following conditions:

$$0 \leqslant x_{r_1}^2 \leqslant z, \quad 0 \leqslant x_{r_2}^2 \leqslant (w-z),$$
$$x_{r_1}^2 + x_{r_2}^2 \in \{x_r^1\}.$$

Using exhaustive search we can obtain all solutions of $\{x_{i_1}^2, x_{i_2}^2\}_{i:=3\ldots k}$.

Suppose that we have a solution for the row with number t. Let the intervals in this step be

$$[1 \ldots j_{t_1}];\ [(j_{t_1}+1) \ldots j_{t_2}];\ldots;\ [(j_{t_{m_t-1}}+1) \ldots w].$$

Let us denote the possibilities for the number of 1's in the different intervals by

$$x_{(t+1)_1}^t, x_{(t+1)_2}^t, \ldots, x_{(t+1)_{m_t}}^t$$

and all solutions for the $(t+1)$th row with fixed $t-1$ rows by $S_{t+1}^{t-1} = \{x_{(t+1)_1}^{t-1}, x_{(t+1)_2}^{t-1}, \ldots, x_{(t+1)_{m_{(t-1)}}}^{t-1}\}$.

If $f = (f_1, f_2, \ldots, f_{m_{(t-1)}}) \in S_{t+1}^{t-1}$ we have the following conditions:

$$0 \leqslant x_{(t+1)_1}^t \leqslant j_{t_1}; \quad 0 \leqslant x_{(t+1)_2}^t \leqslant j_{t_2} - j_{t_1}; \ldots; \quad 0 \leqslant x_{(t+1)_{m_t}}^t \leqslant w - j_{t_{m_t-1}} \tag{1}$$

and

$$x_{(t+1)_{i-1}}^t + x_{(t+1)_i}^t = f_{i'}^s \ \text{ or } \ x_{(t+1)_i}^t = f_{i'} \quad \text{for } i := 1 \ldots m_t \text{ and } \forall f \in S_{t+1}^{t-1}. \tag{2}$$

The last condition depends on whether the interval i' (in step $t-1$) is partitioned in step t (into intervals $i-1$ and i) or not.

Using exhaustive search and these two conditions we can obtain all the sets S_{t+1}^t and then the sets $S_{t+2}^t \ldots S_k^t$.

Similarly we can continue until $t+1=k$.

Comments on efficiency: Let us suppose that in step $k-1$ we have obtained a projective linear $[n, k-1, W]$ code. If we use only condition (1) at the kth step we have to make 2^w checks but if we use conditions (1) and (2) we have to make only $|S_k^{k-1}|$ checks.

In the realization of these ideas we obtain a hierarchic structure of solutions. Each solution in row t is a successor of a solution in row $t-1$. For effective work it is

necessary that the tree of solutions be as small as possible. So it is better to eliminate the equivalent solutions (up to extension).

Let y_1 and y_2 be two solutions for the tth row with corresponding codes C_{y_1} and C_{y_2}. These codes are generated from the first t rows of G_x. y_1 and y_2 are equivalent (up to extension) if $\sigma_1\sigma_2(C_{y_1})=C_{y_2}$ for some permutations σ_1 of the first w coordinates and $\sigma_2 \in \mathrm{Aut}\,(C_0)$.

7. Introduction to the `Split` code

The remainder of the paper consists of the details: what the codes are, why their weights are restricted, why they are classified, and so forth. These details are formulated in the second author's computer language `Split`. Here we will explain only a little bit about how it works, concentrating on what is needed to make some sense out of the details that follow. The reader will probably also find it helpful to consult [15,13]; full details of the `Split` syntax are in [14].

First, what you will see in the remainder of the paper are a sequence of `Split` commands, intermixed with a few comments.[6] These commands must be executed in the specified order. The `Split` program will certify that the assertions implied by the commands are valid. (Were this not the case, the program would immediately terminate, with an error message.) Here is a typical example:

```
[79_7_38] type [79,7,38];
status: weights = {38,40,42,44,46,48,52,54,56,62,64} ;
[base] ⇔ [x1..216] := Unresidue ( [79_7_38], [base], [41_6_19.x1..235] );
status: classified, weights = {38,40,42,44,46,48,54,56,62,64},
    constraints = {mu2 <= 9};
```

We explain what this does. The first line declares that codes with parameters $[79,7,38]$ are under consideration, and attaches the name `[79_7_38]` to the work that follows.

The second line affirms that all such codes have nonzero weights in the given set. This actually invokes a canned procedure to show (via a number of types of linear programming) that there are no other weights.

The third line defines certain codes, to be known for the moment by `[x1]`, `[x2]`, ..., `[x216]`, and outside the current context by `[79_7_38.x1]`, ... These codes are defined by considering residual codes with respect to a word of weight 38. For this, the program accesses the 233 codes with parameters $[41,6,19]$, which were defined previously, and are now referenced by `[41_6_19.x1..235]`. The `[base]` ⇔ part of the line simply encodes the deduction that every $[79,7,38]$ code is obtained in this way.

The last line asserts that the codes (with parameters $[79,7,38]$) have been classified, and establishes (by checking all 216 codes) that no $[79,7,38]$ code has more than 9 words of weight 2 in its dual, and that no such code has a word of weight 52.

[6] A file containing the commands for this paper is distributed with `Split`.

8. Preliminaries

This section contains some Split commands which are preliminary to what follows. First we load the data file from [13]. We refer to the latter paper for the details about the codes and results which appear in the data file. In particular, we are not going to repeat here the definitions of optimal codes which appear in that paper. The remainder of this section consists of some classifications which are prerequisite to the main classifications:

```
accept inputs/code.data1;

[18_5_8] type [18,5,8];
[x44] config from x_8|x44;
[base] ⇔ [x1..39] := Build ( [18_5_8], [x44] );

[17_5_7] type [17,5,7];
[base] ⇔ [a1..95] := [18_5_8.x1..39] – column;

[22_5_10] type [22,5,10];
[x46] config from x_10|x46;
[base] ⇔ [x1..39] := Build ( [22_5_10], [x46] );
status: weights = 10 – {19,21} ;

[21_5_9] type [21,5,9];
[base] ⇔ [a1..183] := [22_5_10.x1..39] – column;

[24_5_11] type [24,5,11];
[base] ⇔ [a1..17] := [25_5_12.{a1..6,b,c} ] - column;

[22_6_9] type [22,6,9];
[base] ⇔ [x1..248] := [23_6_10.a1..29] - column;

[31_6_14] type [31,6,14];
kill y29, y25;

[case1] config 31 ::: {y31 != 0} ;
[y31] config from x_31;
[y31] ⇔ [a1..4]:= Build( [31_6_14], [y31] );

[case2] config 31 ::: {y30 != 0} ;
[y30] config from x_30|x_14_0;
[y30] ⇔ [a5..18]:= Build( [31_6_14], [y30] );

[case3] config 31 ::: {y28 != 0} ;
[y28] config from x_28|x_14_0;
[y28] ⇔ [a19..27] := Build( [31_6_14], [y28] );

[case4] config 31 ::: {y28 = 0, y30 = 0, y31 = 0} ;
[case4] ⇔ [a28..315] := Unresidue ( [31_6_14], [case4], [17_5_7.a1..95] );
```

```
via variable split [base] = [case1] or [case2] or [case3] or [case4];
classification of [base]: [a1..315];
status: classified, weights = {14..18,20,22,24,26,28,30,31} ;

[39_6_18] type [39,6,18];
kill y37, y39 by x_18;

[case1] config 39 ::: {y38 != 0} ;
[y38] config from x_38|x_18_0|x_8_10_0;
[y38] ⇔ [a1..2] := Build( [39_6_18], [y38] );

[case2] config 39 ::: {y36 != 0, y38 = 0} ;
[y36] config from x_36|x_18_0|x990;
[y36] ⇔ [a3..10] := Build( [39_6_18], [y36] );

[case3] config 39 ::: {y36 = 0, y38 = 0} ;
[case3] ⇔ [a11..2035] := Unresidue( [39_6_18], [case3], [21_5_9.a1..183] );

via variable split [base] = [case1] or [case2] or [case3];
classification of [base]: [a1..2035];
status: classified, weights = {18..24,26,28,30,32..34,36,38} ;

[42_6_20] type [42,6,20];
status: weights = {20..22,24,26,28,32,36} ;
[base] ⇔ [x1..35] := Unresidue( [42_6_20], [base], [22_5_10.x1..39] );
[w36_1..2] := Select( [42_6_20.x1..35], {y36 != 0} );

[41_6_19] type [41,6,19];
[base] ⇔ [x1..235] := [42_6_20.x1..35] - column;

[46_6_22] type [46,6,22];
status: weights = {22..26,28,30,..32,36} ;
[base] ⇔ [a1..24] := Unresidue( [46_6_22], [base], [24_5_11.a1..17] );
status: classified, weights = {22..26,28,30..32} ;

[45_6_21] type [45,6,21];
[base] ⇔ [a1..92] := [46_6_22.a1..24] - column;
```

9. An interesting series of codes — the data

```
[8_2_5_with_6] type [8,2,5]{y6 != 0} ;
[a] config from x6|x32;
classification of [base]: [a];
status: unique, enumerator = 1 + 2t^5 + t^6;
```

```
[11_3_6_with_8] type [11,3,6]{y8 != 0} ;
[a] config from x_8|x42|x2211;
classification of [base]: [a];
status: unique, enumerator = 1 + 6t^6 + t^8;

[16_4_8_with_12] type [16,4,8]{y12 != 0} ;
[1] := {1001001111001001,0101001100111010,0011000011111100,
        0000111111111111};
classification of [base]: x_12|x44 ⇒ [1];
status: unique, enumerator = 1 + 13t^8 + 2t^12;
at type [25_5_12];
[w20] := Select( [25_5_12.{a1..6,b,c} ], {y20 != 0} );
```

See also Section 8 for the computation of all $[42,6,20]$ codes, and Section 11 for the computation of all $[75,7,36]$ codes.

10. Optimal codes of dimension 6

```
[32_6_16] type [32,6,16];
[a] := ReedMuller(1,5);
classification of [base]: x_16 ⇒ [a];
status: unique, enumerator = 1 + 62t^16 + t^32;

[38_6_18] type [38,6,18];
[base] ⇔ [x] := P( Even(6) );
status: unique, enumerator = 1 + 30t^18 + 30t^20 + 2t^22 + t^32;

[45_6_22] type [45,6,22];
[base] ⇔ [a] := AntiCode( 6, { {1..4} , {5..6} } );
status: unique, enumerator = 1 + 45t^22 + 15t^24 + 3t^30;

[48_6_24] type [48,6,24];
[a]:= AntiCode( 6, {{1..4} } );
classification of [base]: x_32|x_16_16|x888 ⇒ [a];
status: unique, enumerator = 1 + 60t^24 + 3t^32;

[53_6_26] type [53,6,26];
status: weights = {26,28,30,32} ;
[base] ⇔ [a], [b] := P( [21_5_10.{a,b} ] );
status: classified;
```

The following codes are not optimal, but we need them later:

```
[52_6_25] type [52,6,25];
[base] ⇔ [a1..4] := [53_6_26.{a,b} ] - column;

[56_6_28] type [56,6,28];
[base] ⇔ [a] := AntiCode( 6, {{1..3} } );
```

status: unique, enumerator $=1+56t^{28}+7t^{32}$;

[60_6_30] type [60,6,30];
[base] ⇔ [a] := AntiCode(6, {{1..2} });
status: unique, enumerator $=1+48t^{30}+15t^{32}$;

11. Optimal codes of dimension 7

[40_7_18] type [40,7,18];
credit: existence due to van Tilborg [23];
status: weights = 18_2 - {40} ;

config 38,2 : {10} ;
via lp (joint, iterate⟨y*, div4⟩) [current] = ;
@[base] infer y38 = 0;

[36] config 36,4 : {10} ;
config from x_18_0|x882;
show (common) 53 <= y18 <= 55, 35 < = y20 < = 39, y22 =16,
15 < = y24 < = 19,1 < = y26 < = 3,y36 = 1,27 < = mu3 < = 29, div4 = 56;
at [36];
config from x_14_4|x_5_10_3;
kill x144531, x145431, x145530, x234531, x244530, x245601, x245531, x244631,
x234630 < dual constraint bound = 3, adjoin extra brouwer constraints = 0>;
via lp [current] = ;
@[base] infer y36 = 0;

[base] ⇔ [x1..172]:= Unresidue ([40_7_18], [base], [22_6_9.x1..248]);
status: classified, weights = 18_2 - {36..40} , constraints = {mu2 < =1} ;

[43_7_20] type [43,7,20];
credit: existence due to van Tilborg [21];
status: weights = {20,24,28,32,36} ;
[base] ⇔ [x1..7]:= Unresidue ([43_7_20], [base], [23_6_10,a1..29]);
status: classified, constraints = {84 < = y20 < = 86, 30 < = y24 < = 35,
7 < = y28 < = 10, y32 < = 2, y36 < = 1, y32 + y36 >= 1} ;

[59_7_28] type [59,7,28];
status: weights = {28,30,32,36,40,44,48,52,56} ;
kill y56 by (x_28_0|x_14_14_0);
[base] ⇔ [a1..143]:= Unresidue ([59_7_28], [base], [31_6_14.a1..315]);
status: classified, weights = 28_4- {56} , constraints ={mu2 < = 4} ;

[64_7_32] type [64,7,32];
[base] ⇔ [a] := ReedMuller (1,6);
status: unique, enumerator = $1+126t^{32}+t^{64}$;

[71_7_34] type [71,7,34];
[base] ⇔ [a] := (P (Even(7));
status: unique, enumerator = $1+42t^{34}+70t^{36}+14t^{38}+t^{64}$;

[75_7_36] type [75,7,36];
status: weights = {36,38,40,42,44,48,52,56,60,64,68,72} ;
kill y72 by (x_36_0|x_18_18_0);
[base] ⇔ [a1..3606] := Unresidue([75_7_36], [base], [39_6_18.a1..2035]);
[w68_1..6] := Select([75_7_36.al..3606], {y68 != 0});
status: classified, weights = {36,38,40,42,44,48,52,56,60,64,68} ,
constraints = {mu2 < = 9} ;

[79_7_38] type [79,7,38];
status: weights = {38,40,42,44,46,48,52,54,56,62,64} ;
[base] ⇔ [x1..216] := Unresidue([79_7_38], [base], [41_6_19.x1..235]);
status: classified, weights = {38,40,42,44,46,48,54,56,62,64} ,
constraints = {mu2 < = 9} ;

[82_7_40] type [82,7,40];
status: weights = {40,44,48,56,64} ;
[base] ⇔ [z1..11] := Unresidue([82_7_40], [base], [42_6_20.x1..35]);
status: classified, constraints = {mu2 < = 9} ;

[87_7_42] type [87,7,42];
status: weights = {42,43,44,45,46,48,50,52,54,56,58,60,62,64,70,72} ;
[base] ⇔ [a1..55] := Unresidue([87_7_42], [base], [45_6_21.a1..92]);
status: classified, weights = {42,43,44,45,46,48,50,52,54,58,62,64} ,
constraints = {mu2 < = 3} ;

[90_7_44] type [90,7,44];
status: weights = {44,45,46,48,52,56,60,64,72} ;
[base] ⇔ [a1..6] := Unresidue([90_7_44], [base], [46_6_22.a1..24]);
status: classified, weights = {44,45,46,48,52,60,64} , constraints
= {mu2 < = 3} ;

[93_7_46] type [93,7,46];
[base] ⇔ [a] := AntiCode(7, { {1..5} , {6.. 7} });
status: unique, enumerator = $1+93t^{46}+31t^{48}+3t^{62}$;

[96_7_48] type [96,7,48];
[base] ⇔ [a] := AntiCode(7, { {1..5} });
status: unique, enumerator = $1+124t^{48}+3t^{64}$;

[102_7_50] type [102,7,50];
[base] ⇔ [a1..3] := Unresidue([102_7_50], [base], [52_6_25.a1..4]);
status: classified, weights = {50,52,54,56,58,60,64} ;

[105_7_52] type [105,7,52];
[base] ⇔ [a] := Unresidue([105_7_52], [base], [53_6_26.{a,b}]);
status: unique, enumerator = $1+105t^{52}+15t^{56}+7t^{60}$;

```
[109_7_54] type [109,7,54];
[base] ⇔ [a] := AntiCode( 7, { {1..4} , {5..6} } );
status: unique, enumerator = 1 + 90t^54 + 30t^56 + 6t^62 + t^64;

[112_7_56] type [112,7,56];
[base] ⇔ [a] := AntiCode( 7, {{1..4} } );
status: unique, enumerator = 1 + 120t^56 + 7t^64;

[117_7_58] type [117,7,58];
status: weights = {58,60,62,64} ;
[base] ⇔ [a], [b] := P( [53_6_26.{a,b} ] );
status: classified;

[120_7_60] type [120,7,60];
[base] ⇔ [a] := AntiCode( 7, {{1..3} } );
status: unique, enumerator = 1 + 112t^60 + 15t^64;

[124_7_62] type [124,7,62];
[base] ⇔ [a] := AntiCode( 7, {{1..2} } );
status: unique, enumerator = 1 + 96t^62 + 31t^64;
```

12. Some information on [56,7,26] codes

The construction '[59_ 7_ 28.a1..143] - dual word of weight 3' yields 14466 codes, very quickly. We create a larger set by perturbation, but it takes about 14 h. About 250 more are known — see [14].

```
[56_7_26] type [56,7,26];
kill y49, y53, y45, y41, y39, y37, y55 by (x_26_1, x_27_1, x_26_0),
     y33 by (x_7_22, x_13_18, x_11_22, x_13_20, x_9_20, x_16_14,
     x_16_10), y31 by (x_12_19, x_9_20, x_11_20), y29 by (x_2_27,
     x_0_27, x_3_26, x_4_25, x_5_24, x_7_22, x_8_21, x_14_13,
     x_13_14, x_9_20), y27 by x_12_15;
status : weights = 26_2;
```

```
[a1..19549]:= PertTwo ( [59_7_28.a1..143] - dual word of weight 3 );
```

The following line exhibits all [56,7,26] codes which have disjoint codewords. It is a complete list, although the proof of this fact requires substantial extra work (not shown):

```
[disjoint1..18] :=Select ( [56_7_26.a1..19549], {y52 + y54 + y56 >=
1} );
```

In principle, all [56,7,26] codes could be found by extension, as we have done elsewhere in the paper. One could proceed as follows:

```
[30_6_13] type [30,6,13];
[base] ⇔ [a1..4921] :=[31_6_14.a1..315] - column;
at type [56_7_26];
```

And now a line like

```
[x1..??]:=Unresidue([56_7_26],[base],[30_6_13.a1..4921]);
```

would complete the calculation, but its estimated running time is one year.

Finally, we show that a $[56,7,26]$ code can have no more than 9 words of weight 2 in its dual. Using ordinary linear programming, one only gets the upper bound of 26. Using linear programming based on the joint weight enumerator, one gets the upper bound of 16. We present a much more sophisticated argument, but even so we very much doubt that the outcome is sharp:

```
config 56 ::: {mu2 >=10};
kill y56, y54, y50, y48, y46, jy30y38y42, jy34y34y42, jy34y38y38,
         y52 by x_26_0;
show (joint, iterate) 58 < = y26 < = 80, 21 < = y28 < = 54, y30 < = 20,
         y32 < = 23, y34 < = 22, y36 < = 21, y38 < = 11, y40 < = 5,
         y42 < = 1, y44 < = 1, mu2 < = 16, div4 < = 64;
kill y44 by (x_14_12, x_22_12, x_18_12, x_19_11, x_21_11, x_15_11, x_17_11,
         x_16_10, x_20_10, x_22_10, x_21_9, x_19_9, x_18_8), y42 by (x_18_14,
         x_20_14, x_16_14, x_19_13, x_15_13, x_19_11, x_15_11, x_16_10, x_20_10,
         x_18_10, x_19_9, x_21_9, x_17_9), y38 by (x_16_18, x_18_18, x_19_17,
         x_15_17, x_17_13, x_13_13, x_16_16, x_14_16, x_18_16, x_15_15, x_19_15,
         x_14_14, x_18_14, x_12_14, x_16_14, x_19_13, x_15_13, x_15_11),
         jy30y36y46, jy30y34y40, jy32y36y40, jy34y34y34, jy34y34y40,
         jy34y34y36, jy36y36y36, jy28y40y40, jy30y40y40, jy32y40y40,
         jy34y36y40, jy36y36y40;
show (joint iterate) 64 < = y26 < = 73, 31< = y28 < = 46, y30 < = 9,
         y32 < = 11, y34 < = 11, 6 < = y36 < = 19, 1 < = y40 < = 4,
         mu2 < = 12, div4 >=48;
kill y40 by (x_19_13, x_17_13, x_17_11, x_19_11, x_13_13, x_15_11);
via 1p [current] = ;
@[base] infer mu2 < = 9;
```

Acknowledgements

I. Bouyukliev wishes to thank Prof. Stoyan Kapralov and Dr. Svetlana Topalova. He has used a source code of Prof. Kapralov as a base of his program about isomorphisms of codes in `Extension`. He has also used details from the algorithm of Dr. Topalova on isomorphisms of designs [24].

Appendix: The weight enumerators

[40,7,18]

$1 + 48t^{18} + 57t^{20} + 4t^{22} + 3t^{24} + 12t^{26} + 3t^{28}$

$1 + 51t^{18} + 48t^{20} + 10t^{22} + 9t^{24} + 3t^{26} + 6t^{28}$

$1 + 52t^{18} + 45t^{20} + 13t^{22} + 7t^{24} + 6t^{26} + 3t^{28} + t^{30}$

$1 + 53t^{18} + 42t^{20} + 15t^{22} + 9t^{24} + 3t^{26} + 4t^{28} + t^{30}$

$1 + 53t^{18} + 44t^{20} + 11t^{22} + 9t^{24} + 7t^{26} + 2t^{28} + t^{30}$

$1 + 54t^{18} + 38t^{20} + 18t^{22} + 9t^{24} + 7t^{26} + t^{34}$

$1 + 54t^{18} + 39t^{20} + 20t^{22} + 3t^{24} + 6t^{26} + 5t^{28}$

$1 + 54t^{18} + 40t^{20} + 14t^{22} + 13t^{24} + 2t^{26} + 2t^{28} + 2t^{30}$

$1 + 54t^{18} + 40t^{20} + 17t^{22} + 5t^{24} + 8t^{26} + 2t^{28} + t^{30}$

$1 + 54t^{18} + 42t^{20} + 12t^{22} + 9t^{24} + 6t^{26} + 4t^{28}$

$1 + 55t^{18} + 36t^{20} + 19t^{22} + 10t^{24} + 5t^{26} + t^{30} + t^{32}$

$1 + 55t^{18} + 36t^{20} + 21t^{22} + 7t^{24} + 3t^{26} + 4t^{28} + t^{30}$

$1 + 55t^{18} + 38t^{20} + 14t^{22} + 14t^{24} + 3t^{26} + 2t^{28} + t^{32}$

$1 + 55t^{18} + 39t^{20} + 12t^{22} + 15t^{24} + 3t^{26} + t^{28} + 2t^{30}$

$1 + 56t^{18} + 35t^{20} + 18t^{22} + 10t^{24} + 6t^{26} + t^{28} + t^{32}$

$1 + 56t^{18} + 35t^{20} + 19t^{22} + 9t^{24} + 4t^{26} + 3t^{28} + t^{30}$

$1 + 56t^{18} + 37t^{20} + 13t^{22} + 15t^{24} + 2t^{26} + 3t^{28} + t^{30}$

$1 + 57t^{18} + 27t^{20} + 32t^{22} + 3t^{24} + 6t^{26} + t^{28} + t^{34}$

$1 + 57t^{18} + 34t^{20} + 17t^{22} + 11t^{24} + 5t^{26} + 2t^{28} + t^{30}$

$1 + 57t^{18} + 37t^{20} + 10t^{22} + 15t^{24} + 5t^{26} + 3t^{28}$

$1 + 58t^{18} + 27t^{20} + 30t^{22} + 3t^{24} + 6t^{26} + t^{28} + 2t^{30}$

$1 + 58t^{18} + 33t^{20} + 15t^{22} + 13t^{24} + 6t^{26} + t^{28} + t^{30}$

$1 + 59t^{18} + 27t^{20} + 27t^{22} + 3t^{24} + 9t^{26} + t^{28} + t^{30}$

$1 + 50t^{18} + 51t^{20} + 8t^{22} + 7t^{24} + 6t^{26} + 5t^{28}$

$1 + 52t^{18} + 45t^{20} + 12t^{22} + 11t^{24} + 7t^{28}$

$1 + 52t^{18} + 46t^{20} + 12t^{22} + 10t^{24} + 6t^{28} + t^{32}$

$1 + 53t^{18} + 43t^{20} + 14t^{22} + 7t^{24} + 5t^{26} + 5t^{28}$

$1 + 54t^{18} + 37t^{20} + 22t^{22} + 6t^{24} + 4t^{26} + 3t^{28} + t^{32}$

$1 + 54t^{18} + 38t^{20} + 20t^{22} + 7t^{24} + 4t^{26} + 2t^{28} + 2t^{30}$

$1 + 54t^{18} + 40t^{20} + 13t^{22} + 14t^{24} + 4t^{26} + t^{30} + t^{32}$

$1 + 54t^{18} + 40t^{20} + 16t^{22} + 9t^{24} + 2t^{26} + 6t^{28}$

$1 + 54t^{18} + 41t^{20} + 13t^{22} + 11t^{24} + 4t^{26} + 3t^{28} + t^{30}$

$1 + 54t^{18} + 43t^{20} + 12t^{22} + 3t^{24} + 14t^{26} + t^{28}$

$1 + 55t^{18} + 36t^{20} + 20t^{22} + 9t^{24} + 3t^{26} + 2t^{28} + 2t^{30}$

$1 + 55t^{18} + 37t^{20} + 18t^{22} + 9t^{24} + 5t^{26} + t^{28} + 2t^{30}$

$1 + 55t^{18} + 38t^{20} + 18t^{22} + 5t^{24} + 7t^{26} + 4t^{28}$

$1 + 55t^{18} + 39t^{20} + 14t^{22} + 11t^{24} + 3t^{26} + 5t^{28}$

$1 + 56t^{18} + 35t^{20} + 18t^{22} + 11t^{24} + 4t^{26} + t^{28} + 2t^{30}$

$1 + 56t^{18} + 37t^{20} + 12t^{22} + 16t^{24} + 4t^{26} + t^{28} + t^{32}$

$1 + 56t^{18} + 38t^{20} + 12t^{22} + 13t^{24} + 4t^{26} + 4t^{28}$

$1 + 57t^{18} + 34t^{20} + 16t^{22} + 13t^{24} + 5t^{26} + 2t^{30}$

$1 + 57t^{18} + 36t^{20} + 10t^{22} + 18t^{24} + 5t^{26} + t^{32}$

$1 + 58t^{18} + 26t^{20} + 30t^{22} + 5t^{24} + 7t^{26} + t^{34}$

$1 + 58t^{18} + 28t^{20} + 28t^{22} + 2t^{24} + 10t^{26} + t^{32}$

$1 + 59t^{18} + 26t^{20} + 28t^{22} + 5t^{24} + 7t^{26} + 2t^{30}$

$1 + 60t^{18} + 26t^{20} + 25t^{22} + 5t^{24} + 10t^{26} + t^{30}$

[43,7,20]

$1 + 84t^{20} + 35t^{24} + 7t^{28} + t^{36}$
$1 + 85t^{20} + 33t^{24} + 7t^{28} + 2t^{32}$
$1 + 86t^{20} + 30t^{24} + 10t^{28} + t^{32}$

[59,7,28]

$1 + 78t^{28} + 47t^{32} + t^{36} + t^{52}$
$1 + 80t^{28} + 43t^{32} + 3t^{36} + t^{52}$
$1 + 82t^{28} + 38t^{32} + 6t^{36} + t^{48}$
$1 + 82t^{28} + 39t^{32} + 5t^{36} + t^{52}$
$1 + 84t^{28} + 34t^{32} + 7t^{36} + t^{40} + t^{44}$
$1 + 84t^{28} + 34t^{32} + 8t^{36} + t^{48}$
$1 + 84t^{28} + 35t^{32} + 7t^{36} + t^{52}$
$1 + 85t^{28} + 31t^{32} + 10t^{36} + t^{44}$
$1 + 85t^{28} + 32t^{32} + 7t^{36} + 3t^{40}$
$1 + 85t^{28} + 33t^{32} + 6t^{36} + 2t^{40} + t^{44}$
$1 + 85t^{28} + 33t^{32} + 7t^{36} + t^{40} + t^{48}$
$1 + 85t^{28} + 34t^{32} + 3t^{36} + 5t^{40}$
$1 + 85t^{28} + 35t^{32} + 2t^{36} + 4t^{40} + t^{44}$
$1 + 86t^{28} + 29t^{32} + 10t^{36} + 2t^{40}$
$1 + 86t^{28} + 30t^{32} + 10t^{36} + t^{48}$
$1 + 86t^{28} + 30t^{32} + 9t^{36} + t^{40} + t^{44}$
$1 + 86t^{28} + 31t^{32} + 6t^{36} + 4t^{40}$
$1 + 86t^{28} + 31t^{32} + 8t^{36} + 2t^{44}$
$1 + 86t^{28} + 32t^{32} + 5t^{36} + 3t^{40} + t^{44}$
$1 + 87t^{28} + 27t^{32} + 12t^{36} + t^{44}$
$1 + 87t^{28} + 28t^{32} + 9t^{36} + 3t^{40}$
$1 + 87t^{28} + 29t^{32} + 8t^{36} + 2t^{40} + t^{44}$
$1 + 87t^{28} + 29t^{32} + 9t^{36} + t^{40} + t^{48}$
$1 + 87t^{28} + 30t^{32} + 5t^{36} + 5t^{40}$
$1 + 87t^{28} + 31t^{32} + 4t^{36} + 4t^{40} + t^{44}$
$1 + 88t^{28} + 23t^{32} + 16t^{36}$
$1 + 88t^{28} + 26t^{32} + 11t^{36} + t^{40} + t^{44}$
$1 + 88t^{28} + 26t^{32} + 12t^{36} + t^{48}$
$1 + 88t^{28} + 27t^{32} + 10t^{36} + 2t^{44}$
$1 + 88t^{28} + 27t^{32} + 8t^{36} + 4t^{40}$
$1 + 88t^{28} + 28t^{32} + 7t^{36} + 3t^{40} + t^{44}$
$1 + 88t^{28} + 29t^{32} + 4t^{36} + 6t^{40}$
$1 + 89t^{28} + 22t^{32} + 15t^{36} + t^{40}$
$1 + 89t^{28} + 23t^{32} + 14t^{36} + t^{44}$
$1 + 89t^{28} + 25t^{32} + 10t^{36} + 2t^{40} + t^{44}$
$1 + 89t^{28} + 26t^{32} + 7t^{36} + 5t^{40}$
$1 + 90t^{28} + 22t^{32} + 13t^{36} + t^{40} + t^{44}$
$1 + 90t^{28} + 23t^{32} + 12t^{36} + 2t^{44}$

[79,7,38]

$1 + 50t^{38} + 60t^{40} + 6t^{42} + 10t^{44} + t^{64}$
$1 + 54t^{38} + 46t^{40} + 24t^{42} + 2t^{46} + t^{64}$
$1 + 54t^{38} + 48t^{40} + 18t^{42} + 6t^{44} + t^{64}$
$1 + 56t^{38} + 42t^{40} + 24t^{42} + 4t^{44} + t^{64}$
$1 + 60t^{38} + 30t^{40} + 36t^{42} + t^{64}$
$1 + 62t^{38} + 30t^{40} + 32t^{42} + t^{46} + t^{48} + t^{62}$
$1 + 65t^{38} + 25t^{40} + 30t^{42} + 6t^{44} + t^{62}$
$1 + 67t^{38} + 25t^{40} + 26t^{42} + 6t^{44} + 2t^{46} + t^{62}$
$1 + 76t^{38} + 30t^{40} + 20t^{46} + t^{48}$
$1 + 77t^{38} + 30t^{40} + 18t^{46} + t^{48} + t^{54}$
$1 + 78t^{38} + 28t^{40} + 18t^{46} + 3t^{48}$
$1 + 78t^{38} + 29t^{40} + 18t^{46} + t^{48} + t^{56}$
$1 + 78t^{38} + 30t^{40} + 16t^{46} + t^{48} + 2t^{54}$
$1 + 78t^{38} + 30t^{40} + 17t^{46} + t^{48} + t^{62}$
$1 + 78t^{38} + 30t^{40} + 18t^{46} + t^{64}$
$1 + 79t^{38} + 28t^{40} + 16t^{46} + 3t^{48} + t^{54}$
$1 + 79t^{38} + 29t^{40} + 16t^{46} + t^{48} + t^{54} + t^{56}$
$1 + 79t^{38} + 30t^{40} + 14t^{46} + t^{48} + 3t^{54}$
$1 + 80t^{38} + 28t^{40} + 14t^{46} + 3t^{48} + 2t^{54}$
$1 + 80t^{38} + 28t^{40} + 15t^{46} + 3t^{48} + t^{62}$
$1 + 80t^{38} + 29t^{40} + 14t^{46} + t^{48} + 2t^{54} + t^{56}$
$1 + 81t^{38} + 28t^{40} + 12t^{46} + 3t^{48} + 3t^{54}$

[82,7,40]

$1 + 106t^{40} + 21t^{48}$
$1 + 107t^{40} + 19t^{48} + t^{56}$
$1 + 108t^{40} + 17t^{48} + 2t^{56}$
$1 + 108t^{40} + 18t^{48} + t^{64}$
$1 + 109t^{40} + 15t^{48} + 3t^{56}$
$1 + 90t^{40} + 36t^{44} + t^{64}$
$1 + 92t^{40} + 32t^{44} + 2t^{48} + t^{64}$

[87,7,42]

$1 + 29t^{42} + 48t^{43} + 24t^{44} + 16t^{45} + 7t^{48} + 3t^{58}$
$1 + 33t^{42} + 48t^{43} + 12t^{44} + 16t^{45} + 12t^{46} + 3t^{48} + 3t^{58}$
$1 + 45t^{42} + 72t^{44} + 7t^{48} + 3t^{58}$
$1 + 49t^{42} + 60t^{44} + 12t^{46} + 3t^{48} + 3t^{58}$
$1 + 50t^{42} + 54t^{44} + 20t^{46} + 2t^{54} + t^{64}$
$1 + 50t^{42} + 60t^{44} + 6t^{46} + 8t^{48} + 2t^{52} + t^{64}$

$1 + 51t^{42} + 54t^{44} + 18t^{46} + t^{48} + 3t^{58}$
$1 + 53t^{42} + 56t^{44} + 8t^{46} + 7t^{48} + 3t^{58}$
$1 + 54t^{42} + 48t^{44} + 18t^{46} + 4t^{48} + 2t^{52} + t^{64}$
$1 + 54t^{42} + 50t^{44} + 14t^{46} + 4t^{48} + 4t^{50} + t^{64}$
$1 + 54t^{42} + 54t^{44} + 2t^{46} + 16t^{48} + t^{64}$
$1 + 56t^{42} + 42t^{44} + 24t^{46} + 2t^{48} + 2t^{52} + t^{64}$
$1 + 56t^{42} + 44t^{44} + 20t^{46} + 2t^{48} + 4t^{50} + t^{64}$
$1 + 56t^{42} + 46t^{44} + 14t^{46} + 8t^{48} + 2t^{50} + t^{64}$
$1 + 57t^{42} + 44t^{44} + 20t^{46} + 3t^{48} + 3t^{58}$
$1 + 58t^{42} + 40t^{44} + 20t^{46} + 6t^{48} + 2t^{50} + t^{64}$
$1 + 58t^{42} + 42t^{44} + 14t^{46} + 12t^{48} + t^{64}$
$1 + 58t^{42} + 44t^{44} + 14t^{46} + 8t^{48} + 2t^{52} + t^{64}$
$1 + 60t^{42} + 36t^{44} + 20t^{46} + 10t^{48} + t^{64}$
$1 + 60t^{42} + 38t^{44} + 20t^{46} + 6t^{48} + 2t^{52} + t^{64}$
$1 + 60t^{42} + 40t^{44} + 16t^{46} + 6t^{48} + 4t^{50} + t^{64}$
$1 + 60t^{42} + 42t^{44} + 10t^{46} + 12t^{48} + 2t^{50} + t^{64}$
$1 + 61t^{42} + 40t^{44} + 16t^{46} + 7t^{48} + 3t^{58}$
$1 + 61t^{42} + 48t^{44} + 15t^{48} + 3t^{58}$
$1 + 62t^{42} + 28t^{44} + 32t^{46} + 2t^{48} + 2t^{50} + t^{64}$
$1 + 62t^{42} + 36t^{44} + 16t^{46} + 10t^{48} + 2t^{50} + t^{64}$
$1 + 62t^{42} + 38t^{44} + 10t^{46} + 16t^{48} + t^{64}$
$1 + 64t^{42} + 28t^{44} + 27t^{46} + 3t^{48} + 4t^{50} + t^{62}$
$1 + 64t^{42} + 28t^{44} + 28t^{46} + 2t^{48} + 4t^{50} + t^{64}$
$1 + 65t^{42} + 25t^{44} + 29t^{46} + 5t^{48} + t^{50} + t^{52} + t^{62}$
$1 + 65t^{42} + 28t^{44} + 28t^{46} + 3t^{48} + 3t^{58}$
$1 + 65t^{42} + 36t^{44} + 12t^{46} + 11t^{48} + 3t^{58}$
$1 + 66t^{42} + 24t^{44} + 27t^{46} + 7t^{48} + 2t^{50} + t^{62}$
$1 + 67t^{42} + 25t^{44} + 25t^{46} + 5t^{48} + 3t^{50} + t^{52} + t^{62}$
$1 + 68t^{42} + 24t^{44} + 23t^{46} + 7t^{48} + 4t^{50} + t^{62}$
$1 + 69t^{42} + 24t^{44} + 24t^{46} + 7t^{48} + 3t^{58}$

[90,7,44]

$1 + 45t^{44} + 64t^{45} + 15t^{48} + 3t^{60}$
$1 + 69t^{44} + 48t^{46} + 7t^{48} + 3t^{60}$
$1 + 77t^{44} + 32t^{46} + 15t^{48} + 3t^{60}$
$1 + 90t^{44} + 34t^{48} + 2t^{52} + t^{64}$
$1 + 92t^{44} + 30t^{48} + 4t^{52} + t^{64}$
$1 + 93t^{44} + 31t^{48} + 3t^{60}$

References

[1] B.I. Belov, A conjecture on the Griesmer boundary, in: Optimization Methods and Their Applications (All-Union Summer Sem., Khakusy, Lake Baikal, 1972) (Russian), Sibersk. Ènerget. Inst. Sibirsk. Otdel. Akad. Nauk SSSR (Irkutsk), 1974, pp. 100–106, 182.

[2] I. Bouyukliev, D.B. Jaffe, V. Vavrek, The smallest length of eight-dimensional binary linear codes with prescribed minimum distance, IEEE Transactions on Information Theory 46 (2000) 1539–1544.

[3] A.E. Brouwer, The uniqueness of the binary linear [27,7,12] code, accessible over the World Wide Web via `http://www.win.tue.nl/~aeb/preprints/Unique.27.7.12.gz`, 1992.

[4] A.E. Brouwer, A.M. Cohen, A. Neumaier, Distance-Regular Graphs, Springer, New York, 1989.

[5] A.E. Brouwer, M. van Eupen, The correspondence between projective codes and 2-weight codes, Designs Codes Cryptogr. 11 (1997) 261–266.

[6] S.M. Dodunekov, The minimum block length of a linear q-ary code with given dimension and code distance, Problemy Peredachi Informatsii 20 (1984) 11–22 (in Russian).

[7] S.M. Dodunekov, Optimal linear codes, Ph.D. Thesis, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1985.

[8] S.M. Dodunekov, N.L. Manev, Characterization of two classes of codes that attain the Griesmer bound, Problems Inform. Transmission 19 (1983) 253–259.

[9] T. Helleseth, Further classifications of codes meeting the Griesmer bound, IEEE Trans. Inform. Theory 30 (1984) 395–403.

[10] T. Helleseth, A characterization of codes meeting the Griesmer bound, Inform. and Control 50 (1981) 128–159.

[11] T. Helleseth, Projective codes meeting the Griesmer bound, Discrete Math. 106/107 (1992) 265–271.

[12] R. Hill, Caps and codes, Discrete Math. 22 (1978) 111–137.

[13] D.B. Jaffe, Optimal binary linear codes of length $\leqslant 30$, Discrete Math. 223 (2000) 135–155.

[14] D.B. Jaffe, Binary linear codes: new results on nonexistence, preprint (ongoing work), Version 0.5 (11/10/97), accessible over the World Wide Web via `http://www.math.unl.edu/~djaffe/codes/code.ps.gz` or `code.dvi.gz`; see .../~djaffe/binary/codeform.html for an online database, which is more frequently updated.

[15] D.B. Jaffe, in: T. Mora, H. Mattson (Eds.), A Brief Tour of Split Linear Programming, Proceedings of the AAECC 12, Lecture Notes in Computer Science, Vol. 1255, Springer, New York, 1997, pp. 164–173.

[16] S.N. Kapralov, Enumeration of the binary linear [24,7,10] codes, Proceedings of the Fifth International Workshop on Algebraic and Combinatorial Coding Theory, Unicorn, Shumen, Bulgaria, 1996, pp. 151–156.

[17] B.D. McKay, Practical graph isomorphism, Congr. Numer. 30 (1981) 45–87.

[18] B.D. Mckay, Nauty user's guide (Version 1.5), 1990, This guide as well as the nauty source files are available over the World Wide Web via `http://cs.anu.edu.au/people/bdm/nauty19/nauty19p.tar.Z`.

[19] E.M. Rains, N.J.A. Sloane, Self-dual codes, in: V.S. Pless, W.C. Huffman (Eds.), Handbook of Coding Theory, Elsevier Science, Amsterdam, 1998.

[20] G. Solomon, J.J. Stiffler, Algebraically punctured cyclic codes, Inform. and Control 8 (1965) 170–179.

[21] H. van Tilborg, On quasi-cyclic codes with rate $1/m$, IEEE Trans. Inform. Theory 24 (1978) 628–630.

[22] H. van Tilborg, On the uniqueness resp. nonexistence of certain codes meeting the Griesmer bound, Inform. and Control 44 (1980) 16–35.

[23] H. van Tilborg, The smallest length of binary 7-dimensional linear codes with prescribed minimum distance, Discrete Math. 33 (1981) 197–207.

[24] S. Topalova, Construction and investigation of combinatorial designs with given automorphisms, Ph.D. Thesis, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 1998.